VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Metodiky popisu při vývoji SW díla
Description Methods for Software Development Process

2012 Bc. Zdeněk Špunda

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Zdeněk Špunda**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Metodiky popisu při vývoji SW díla
Description Methods for Software Development Process

Zásady pro vypracování:

Cílem této práce je vytvořit metodiku pro implementaci softwarového procesu vycházející z kombinace stávajících agilních a robustních metodik. Metodika bude zohledňovat současné aktuální potřeby vývoje SW.
Práce bude obsahovat zejména:
1. Nalezení dostupných a používaných metodik a jejich popis.
2. Roztřídění používaných metodik do skupin dle charakteristických společných znaků.
3. Porovnání metodik.
4. Použití kombinace metodik na konkrétních příkladech.
5. Doporučení pro modelové případy vývoje software.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Svatopluk Štolfa, Ph.D.**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

„Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal."

V Ostravě 3.5.2012 Podpis studenta .....

Abstrakt

Závěrečná diplomová práce se zabývá metodikami vývoje softwarového díla. Úvodní kapitoly se snaží především stručně popsat vybrané metodiky a stanovit kritéria pro výběr vhodné metodiky. Další kapitoly se zabývají technikami, jak zlepšit vývojový proces kombinacemi stávajících robustních a agilních metodik a jejich použitím na modelových příkladech.

Klíčová slova

metodika vývoje, RUP, Scrum, XP, ASD, OpenUP, Lean Development, DSDM, MSF, výběr vhodné metodiky vývoje

Abstract

The final thesis deals with methods of software development work. Introductory chapters try to briefly describe selected methods and establish criteria for selecting the appropriate methodology. Other chapters deal with techniques to improve the development process combinations of existing robust and agile methodologies and their application to model examples.

Keywords

development methodology, RUP, Scrum, XP, ASD, OpenUP, Lean Development, DSDM, MSF, selection of appropriate methodology

Seznam použitých symbolů a zkratek

Aj. – a jiné
Ad. – a další
ASD – Adaptive Software Development
Atd. – A tak dále
DSDM – Dynamic Software Development Method
EPF – Eclipse proces Framework
FDD – Feature Driven Development
IBM – International Business Machines Corporation (společnost na trhu informačních technologí)
MSF – Microsoft Solution Framework
Např. – Například
RUP – Rational Unified Process
SW – software
UML – Unified Modeling Language (grafický jazyk pro vizualizaci, specifikaci, navrhování a dokumentaci programových systémů)
UP – Unified Proces
XP – Extreme Programming

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

Oblast vývoje software a obecně softwarového inženýrství se neustále rozvíjí a buduje. Důvodem jsou nové požadavky kladené na vyvíjený software. V dnešní době existuje mnoho metodik, jak zefektivnit vývojový proces. Při vývoji se do roku 2000 používali především tradiční metody, avšak potom se začínají rozšiřovat pod souhrnným označením agilní metodiky.

Agilní metodiky vycházejí z Manifestu agilního vývoje (Manifesto for Agile Software Development), který byl sepsán 17 vývojáři v roce 2001. Obsahuje základní principy, jak je možné vyvíjet software efektivněji, rychleji a kvalitněji.

Hlavním cílem diplomové práce je vytvořit metodiku pro implementaci softwarového procesu vycházející z kombinace stávajících agilních a robustních metodik. Metodika bude zohledňovat současné aktuální potřeby vývoje SW.

1.1 Struktura práce

Diplomová práce je členěna do sedmi kapitol. První kapitole uvádím práci a popsuji její cíl. Druhá kapitola stručně popisuje dostupné a používané metodiky ve vývoji SW, jejich základní principy. Třetí kapitola diplomové práce stanovuje kritéria výběru vhodné metodiky na základě charakteristických znaků. Ve Čtvrté kapitole vybraným metodikám přiřazuji hodnocení kritériím stanoveným v předchozí kapitole. Pátá kapitola popisuje jak si vytvořit metodiku a použití kombinace metodik na konkrétních příkladech. Šestá kapitola popisuje vytvořený SW pro doporučení vhodné metodiky vývoje. Poslední kapitola je závěrem práce, kde se snažím zhodnotit dosažené výsledky.

2 Popis dostupných a používaných metodik

Než se pustím do popisu vybraných metodik, popíši co to vlastně metodika je. Metodika je označení komplexních postupů a návodů, jak vyvíjet softwarové aplikace. Nezabývá se konkrétním způsobem řešení (jak provést analýzu, v jakém programovacím jazyce má být SW implementován), ale zabývá se spíše pohledem z výšky. Řeší otázky Proč? Kdo? Kdy? Co? Nemusí rozebírat jak danou operaci provést. [1]

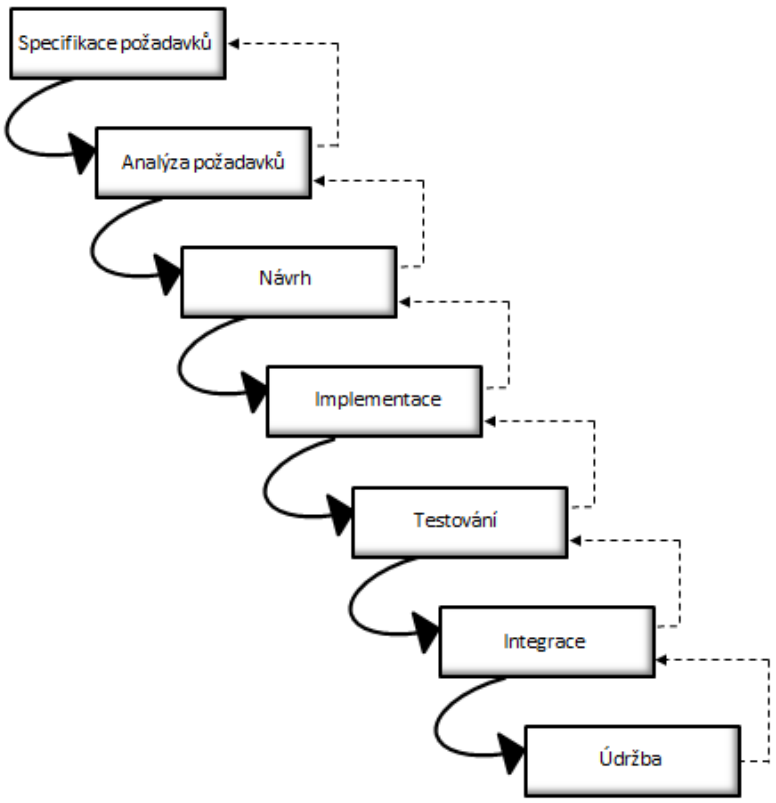
2.1 Vodopádový model

Vodopádový model vznikl v sedmdesátých letech minulého stolení. Nejedná se úplně o metodiku, ale model životního cyklu. Je považován za základ při vývoji SW, vnáší do vývoje disciplínu. Existují různé rozšíření vodopádového modelu, které se snaží řešit nevýhody původního vodopádového modelu. Jako příklad jsem uvedl upravený vodopádový model s možností návratnosti do předchozích fází na obrázku číslo 1.

Celý proces se skládá ze sedmi částí. Začíná fází specifikace požadavků, poté následují fáze analýza požadavků, návrh, implementace, testování, integrace a poslední fází je provoz a údržba aplikace.

Princip spočívá v postupném plnění jednotlivých fází. Fáze nemůže začít, dokud předchozí fáze nebyla schválena. Při dokončení každé fáze, můžeme provést kontrolu dosaženého stavu, a v případě nedostatků se můžeme vrátit do předchozí fáze.

Vodopádový model je vhodný pro malé a velmi vyspecifikované projekty, protože dodavatel potřebuje zákazníka pouze na začátku projektu ve fázi specifikace požadavků, a pak na konci při předání hotového programového díla. Z toho plyne řada nedostatků. Vývoj je nepružný. Postupem fází vodopádového modelu vzniká riziko neúspěšnosti projektu. Zákazník kolikrát neví na začátku projektu, co přesně chce, nebo se můžou jeho požadavky změnit a při předání zjistí, že tohle vlastně nechtěl. Tím dochází ke zpoždění projektu, protože všemi fázemi musíme projít znovu. Zákazník nemůže v průběhu implementace ovlivňovat uživatelské rozhraní ani jeho funkcionalitu. Výhodou je jednoduchost, ale ta platí pouze u malých projektů.



Obrázek 1: Vodopádový model

2.2 Rational Unified Process (RUP)

Metodika Rational Unified Process původně vlastněna Rational Corporation, kterou v roce 2003 převzala společnost IBM. Do roku 1999 byla považována za komerční implementaci metodiky UP. Dnes bychom měli považovat UP za otevřený standart a RUP za komerční specifickou podtřídu UP. RUP se neustále vyvíjí a vylepšuje, není to pouze metodika, ale nabízí množství nástrojů a aplikací. Pro každý z nástrojů poskytuje tzv. rádce Tool Mentors.

Metodika RUP je rozsáhlá, obecná, a propracovaná. Je vhodná spíše pro větší projekty velké organizace a vývojové týmy, ale díky obecnosti jde použít pro široké spektrum projektů. U menších týmů a rozsáhlých projektů, může být vývoj podle metodiky neefektivní. Tým může strávit příliš času zkoumáním metodiky, vytvářením meziproduktů, dokumentů a jejich formálních náležitostí. Mezi výhody metodiky patří provázanost s UML notací, objektový přístup, klade důraz na práci s riziky, na jejich vyhledávání, analýzu a řízení. Dále lze zařadit mezi výhody podvědomí o metodice a její uznání. Značnou nevýhodou pro malé vývojové týmy je fakt, že RUP je komerční produkt a licence je drahá.

Základní principy

- Softwarový produkt je vyvíjen iterativně

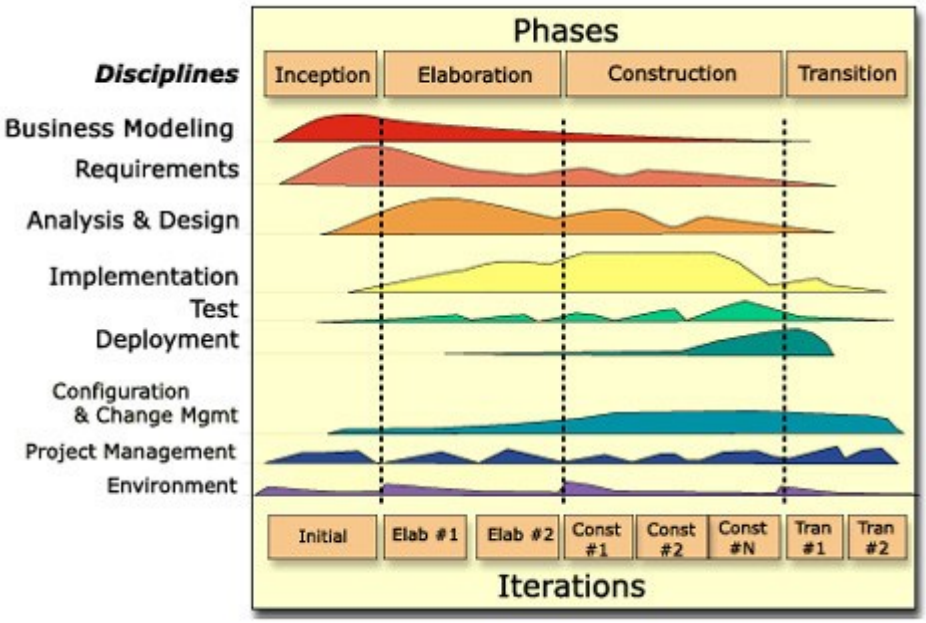
- Jsou spravovány a řízeny požadavky na něj kladené
- Využívá se již existujících komponent
- Model softwarového systému je vizualizován
- Průběžné ověřování a zajišťování kvality
- Změny systému jsou řízeny

Na obrázku č. 2 je znázorněn vývojový cyklus skládající se ze 4 fází.

Fáze vývojového cyklu:

- Inception (Zahájení)
- Elaboration(Projektování)
- Construction (Realizace)
- Transition (Předání)

Fáze realizace zabírá nejvíce času, udává se v průměru 50% z celého projektu. Každá fáze může mít různý počet iterací. Stanovit přesnou dobu iterace není možné, záleží na velikosti projektu, počtu pracovníků a jejich schopnostech a zkušenostech a v neposlední řadě na celkovém plánu projektu.



Obrázek 2: Vývojový cyklus v RUP [6]

RUP definuje 4 základní typy elementů, odpovídající na základní otázky

- Pracovníci (workers) – řeší otázku Kdo?
- Činnosti (activities) – řeší otázku Jak?
- Meziprodukty (artifacts) – řeší otázku Co?
- Pracovní procesy (workflows) – řeší otázku Kdy?

2.3 Extreme programming

Metodika XP Extrémního programování je zástupcem agilních metodik. Vznik se datuje do roku 1999, kdy Kent Beck představil jeho základní myšlenky. Patří mezi nejznámější agilní metodiky, vhodnou pro malé až střední týmy typicky 2 až 10 programátorů. XP nominuje psaní zdrojového kódu na klíčovou činnost při vývoji SW. Jak uvádí Kent Beck ve své knize [3] XP je lehký, účinný, nepříliš rizikový, pružný, předvídatelný, vědecký a zábavný způsob, jak vyvíjet software.

2.3.1 Nejdůležitější hodnoty XP

Komunikace

Komunikace v projektu je důležitým aspektem. Nesdělením důležitých změn v projektu někomu jinému vede k problémům, proto XP používá postupy, jak komunikaci udržovat jde např. o testování jednotek, párové programování, odhadování úkolů atd. Pokud dochází k výpadkům komunikace XP definuje kouče, který má za úkol navazovat jejich vztahy.

Jednoduchost

Další hodnotou je jednoduchost. Metodika předpokládá, že lepší udělat jednoduchou věc dnes, a pokud je potřeba zítra ji upravit, než udělat složitou věc hned. Zákazník může změnit své požadavky a složitá implementace nemusí být nikdy použita. XP se zaměřuje na to, co je důležité pro zákazníka nyní.

Zpětná vazba

Jednou z nejdůležitějších metod získání zpětné vazby je testování. Programátoři píší jednotkové testy a tím získávají okamžitou zpětnou vazbu napsaného kódu, zda pracuje jak má. Dále pak testeři a zákazníci píší testy funkcionality. Zákazníci sledují v průběhu projektu aktuální stav, zda odpovídá časovému. Zpětná vazba úzce souvisí z komunikací.

Odvaha

Budeme říkat pravdu o pokroku, nebojíme se změn v požadavcích a návrhu. Pokud z nějakého důvodu nevede cesta dopředu, musí se opravit za jakoukoli cenu. Dokonce i v případě, že by to znamenalo zahodit dvě třetiny zdrojového kódu.

2.3.2 Dvanáct základních postupů XP

Kent Beck definoval 12 konkrétních postupů, které by měli vést k vytváření kvalitních softwarových produktů.

Plánovací hra

Plánovací hry se účastní vedení i se zákazníkem a vývojový tým. Cílem plánovací hry je, aby zákazník mohl ovlivňovat vývoj a směr projektu. Plánovací hra se skládá ze tří fází průzkumu, závazku a řízení.

- V průzkumu zákazník napíše uživatelské příběhy (User Stories), vývojový tým odhadne čas, jak dlouho bude trvat implementace. Pokud nemůže odhadnout dobu potřebnou k implementaci, požádá o vyjasnění nebo rozdělení zadání.
- Účelem fáze závazku je, aby vedení zvolilo šířku zadání, datum uvolnění další verze a vývoj se zavázal k jejímu vyvinutí.
- Účelem fáze řízení je aktualizace plánu podle dosavadních poznatků a běhu vývoje.

XP definuje ještě podrobnější typ plánování iterační plánovací hru, která plánuje typicky jeden až tři týdny.

Malé verze

XP doporučuje vydávat malé verze, díky neustále integraci. Neustále testování snižuje chybovost, proto po vydání verze není nutno dlouho testovat.

Metafora

Metafora je to vlastně vize projektu, pomáhá všem v pochopení základních prvků a vztahů mezi nimi. Důležité je, aby všichni používali stejnou terminologii.

Jednoduchý návrh

Systém má být navrhován co nejjednodušeji. Obsahovat jenom to co je nutné pro splnění daného cíle.

Testování

Nepřetržité testování programátoři píší jednotkové testy, zákazník píše testy funkcionality. Jakákoliv funkce bez automatického testu by zkrátka neměla existovat.

Re faktorizace

Re faktorizace znamená úpravu a zjednodušení stávajícího programu bez změny chování.

Párové programování

Vývoj probíhá v párech u jednoho počítače. Jeden implementuje a druhý přemýšlí o dopadech na testy, jaké další testy napsat, dále o možnosti zjednodušení implementace. Tyto páry jsou dynamické.

Společné vlastnictví

V XP může kdokoli provést jakoukoliv změnu ve zdrojovém kódu. Na základě neustále se měnících se párů mají všichni přehled o všech částech programu.

Nepřetržitá integrace

Zdrojový text se integruje a testuje několikrát denně. Každý je zodpovědný za svůj test a musí ho opravit. Po provedení testování musí být všechny testy úspěšné.

Čtyřicetihodinový pracovní týden

XP říká, že není možné odpracovat mnohokrát za sebou přesčas a být stále svěží a kreativní. Přesčas je příznakem problému.

Zákazník na pracovišti

Součástí týmu se po dobu vývoje stává skutečný uživatel nebo zákazník. Tedy člověk, který se systémem bude pracovat. Odpovídá na otázky, řeší spory a určuje priority.

Standarty pro psaní zdrojového textu

Protože v podstatě nevzniká žádná dokumentace a specifikace a zdrojový kód je základním nositelem informací je nutné se domluvit na standardech pro dodržování psaní zdrojového kódu, pro čitelnost a přehlednost.

2.3.3 Role týmu

XP se zaměřuje na práci s lidmi, složením vývojového týmu, definuje lidské role, motivaci lidí a dokonce uspořádáním pracovišť.

Programátor

Náplní práce je programování, design, re faktorizace a testování po jednotkách,

Zákazník

Uživatel systému, zná jak má systém vypadat, má možnost ovlivňovat projekt

Tester

Pomáhání zákazníkovi při psaní testů funkcionality, testy uživatelského rozhraní, spouští všechny testy a zajišťuje funkčnost všech testovacích nástrojů,

Stopař

Provádí časové odhady, analyzování zpoždění projektu, celkový dohled nad projektem.

Kouč

Odpovědný za proces jako celek.

Konzultant

Poskytuje hluboké znalosti o nějaké problematice.

XP probíhá v iteracích, je vhodné pro projekty s často se měnícími požadavky nebo nejasným zadáním. Není vhodné pro velké programátorské týmy a v případech, kdy zákazník nebo vedení trvá na kompletní specifikaci, analýze nebo návrhu před programováním.

2.4 Scrum Development Process

Metodika Scrum Develoment Process patří do skupiny agilních metodik. Metodika byla poprvé představena a publikována na konferenci v roce 1995 Kenem Schwaber a Jeff Sutherland. Název Scrum vznikl z anglického slova, jehož význam spadá do ragby a označuje skrumáž, mlýn. Jak popisuje [9], Scrum je rámec, který umožňuje aplikovat různé procesy a techniky. Úlohou je zviditelnění relativní efektivnosti Vašich vývojových postupů, tak abyste se mohli zlepšovat, a poskytuje Vám rámec pro vývoj komplexních produktů.

Metodika Scrum je vhodná i pro velmi důležité a rozsáhle projekty. Scrum se zabývá řízením celého vývojového procesu a ne implementačními postupy. Vývoj probíhá iterativně a klade důraz na řízení rizik. Metodika dokáže pružně reagovat na změny.

2.4.1 Časové rámce

Na obrázku 3 je znázorněn Scrum proces v rámci jednoho sprintu. Sprint trvá 2-4 týdny. Scrum využívá časové rámce (Time-Boxes) k vytvoření pravidelnosti, dokončení potenciálně nasaditelné verze.

Release Planning Meeting (Meeting pro plánování release)

Plán vydání obsahuje cíl vydání, nejdůležitější části produktového backlogu, seznam největších rizik a přehled vlastností a funkcí, které budou součástí vydání. Také stanovuje pravděpodobné datum dokončení a předpokládané náklady. Plán se může měnit mezi jednotlivými sprinty.

Sprint Planning Meeting (Plánovací meeting pro Sprint)

Meeting se skládá ze dvou částí. V první se rozhoduje, co tým bude dělat v rámci sprintu, definuje se cíl, a ve druhé části, se řeší, jak bude tým postupovat, návrh práce a identifikování úkolů.

Sprint (Samotný vývoj)

Sprint zahrnuje plánování sprintu, fázi vývoje, review meetingu a retrospektivy.

Daily Scrum

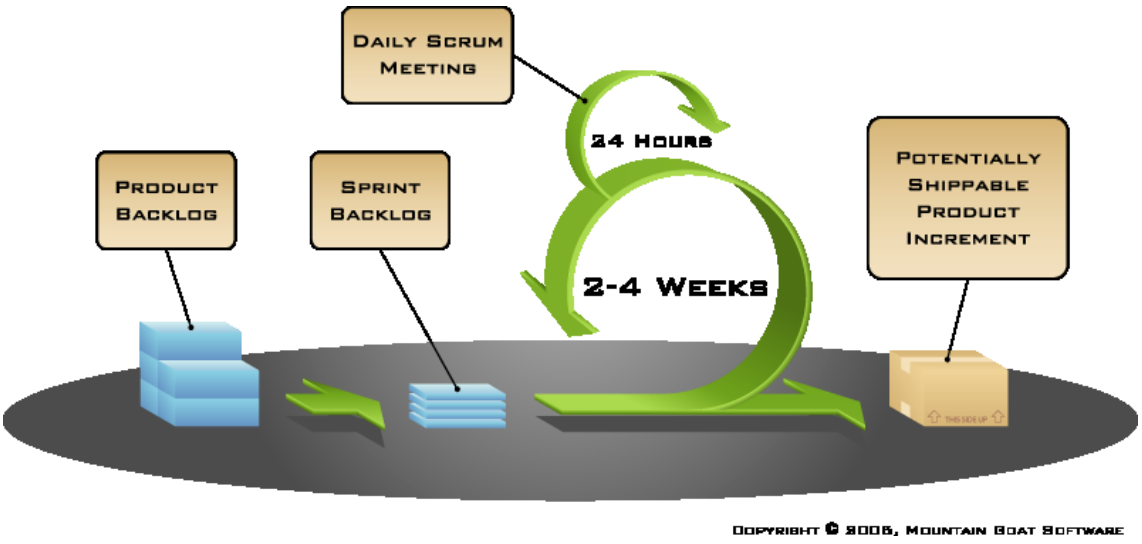
Každý člen týmu popíše, co dokončil od posledního meetingu, co hodlá dělat před dalším meetingerm a jaké překážky mu v tom brání.

Sprint Review (zhodnocení)

Produkt Owner odliší, co bylo, a co nebylo uděláno. Tým prodiskutuje, co šlo dobře, na jaké problémy narazil a co by šlo vylepšit. Je to cenný výstup pro plánování dalšího sprintu.

Sprint Retrospective (Retrospektiva)

Shodnocení minulého sprintu, co by šlo zlepšit, čeho byste se měli vyvarovat, atd.



Obrázek 3: Scrum proces [10]

2.4.2 Role týmu

Scrum definuje tři role v rámci týmu Scrum Master, Team a Produkt Owner. Tým se organizuje sám, neobsahuje žádného šéfa, který by říkal, kdo jaký úkol bude plnit.

Scrum Master

Zodpovídá za dodržování metodiky, pomáhá týmu v adaptaci Scrumu, má za úkol odstraňování překážky. Úlohou Scrum Mastera v týmu je být podpůrným vedoucím.

Produkt Owner (Vlastník produktu)

Vlastník produktu je jedinou osobou zodpovědnou za Product Backlog (seznam požadavků) a zajišťuje, aby byl všem přístupný a všichni věděli, na čem mají pracovat. Definuje funkce (v souladu s vizí), prioritizuje funkce.

Team

Cílem týmu je, aby v rámci Sprintu přeměnili Product Backlog v přírůstek potenciálně nasaditelné funkčnosti, tým dále odhaduje náročnost práce.

2.4.3 Artefakty

Artefakty Scrumu zahrnují Product backlog, release burn down, sprint backlog a sprint burndown.

Product backlog

Product backlog obsahuje seznam požadavků na vyvíjený produkt, je dynamický a mění se podle potřeb. Položky mají popis, prioritu a odhad. Jsou seřazeny podle priority.

Release burn down

Graf zaznamenává součet zbývajícího odhadovaného úsilí v čase.

Sprint backlog

Skládá se ze seznamu úkolů. Úkoly musí být dekomponovány, aby mohla být sledováno plnění úkolu na Daily Scrumu, délka úkolu jeden den a méně.

Sprint burndown

Graf, který ukazuje množství práce, která v průběhu času zbývá k dokončení sprintu.

2.5 Open UP

Open UP je open-source a je součástí Eclipse Process Frameworku (EPF). Open UP patří do agilních metodik. Je to minimální dostatečná, kompletní metodika vývoje software, která je snadno přizpůsobitelná a rozšiřitelná. Metodika má základní vlastnosti z UP (Unified Process), je tedy založena na inkrementálním modelu životního cyklu, případech užití, řízení rizik a architektuře.

2.5.1 Základní zásady Open UP

Popis čtyř základních zásady vývoje podle Open UP.

Spolupráce pro sladění zájmů a sdílení porozumění

Prosazovat postupy, které podporují zdravé týmové prostředí, spolupráci a rozvíjet společné porozumění projektu.

Hodnocení priorit s cílem maximalizovat hodnoty pro zákazníka

Tento princip podporuje postupy, které umožní všem účastníkům projektu k rozvoji řešení s cílem maximalizovat hodnotu pro zákazníka.

Včasné zaměření na architekturu s cílem minimalizace rizik

Vývoj zaměřit na architekturu, snížit rizika a vyvíjet organizovaněji.

Průběžně získávat zpětnou vazbu a zlepšovat

Tento princip podporuje postupy, které umožňují týmu získávat brzkou zpětnou vazbu od zákazníka a také předvést přidanou hodnotu produktu.

2.5.2 Vrstvy Open UP

Proces podle Open Up se skládá ze tří vrstev, jak je vidět na obrázku č. 4. Micro-Increment, Iteration Lifecycle a Project Lifecycle.

Micro-Increment (Mikro-přírůstek)

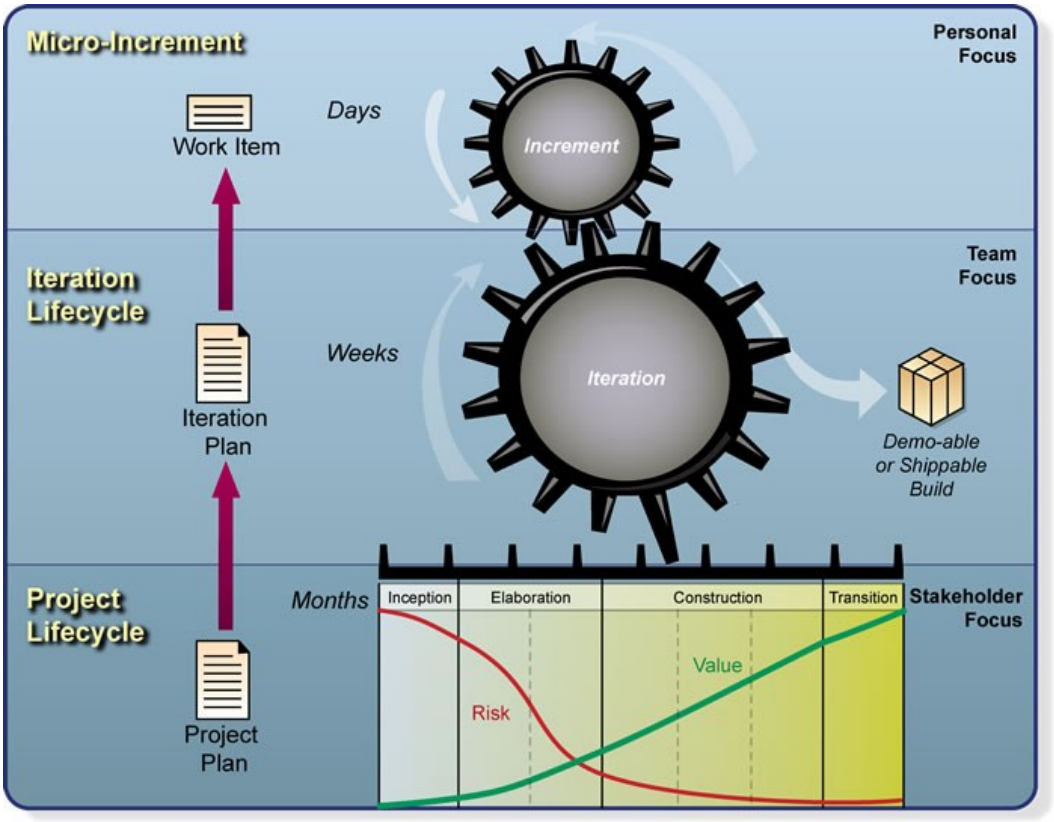
Mikro-přírůstek představuje výsledek několika hodin až dní práce jedné osoby (nebo spolupracujících osob), aby se dosáhlo cíle iterace. Mikro přírůstky poskytují velmi krátkou zpětnou vazbu, která řídí adaptivní rozhodování v během iterace.

Iteration Lifecycle (Iterace životního cyklu)

Iterace životního cyklu jsou měřené v týdnech, jejímž výsledkem je o testovatelná a spustitelná verze. Většinu času se tráví plněním Mikro-inkrementů. Na konci se provádí zhodnocení, jak zlepšit proces pro další iterace.

Project Lifecycle (Životní cyklus projektu)

Je rozdělen do 4 fází Zahájení, Projektování, Realizace a Předání. Každá fáze končí milníkem, jehož cílem je, poskytnou dohled nad projektem.



Obrázek 4: Vrstvy Open UP

2.5.3 Role týmu

Open UP definuje tyto role: analytik (analyst), architekt(architect), vývojář(developer), tester, zúčastněné strany (stateholders), vedoucí projektu (project manager) a kdokoli (any role).

2.6 Lean Development

Metodika Lean Development je zástupcem agilních metodik. Metodika vychází z myšlenek výrobního procesu Lean Manufacturing, který vznikl v 80. letech. Cílem metodiky je vyvíjet SW rychleji s menšími náklady a uspokojení zákazníkových požadavků. K tomu využívá deset základních principů. [12]

2.6.1 Základní pravidla

Popis deseti pravidel používaných ve vývoji SW podle Lean Development.

Odstranit zbytečnosti

Odstranit vše, co nepřináší hodnotu produktu a zákazník to nevyžaduje např. dokumenty, diagramy, nebo modely.

Minimalizovat zásoby (artefakty)

Vytvářet pouze artefakty, které jsou podle předchozího pravidla užitečné a zaměřit se na rozsah zpravování. Nevytvářet dlouhé a detailní specifikace.

Maximalizovat tok

Zkrátit čas vývoje, vývoj probíhá iterativně.

Vývoj tažený poptávkou

Dodávku přizpůsobit požadavků zákazníka. Všechna rozhodnutí učinit až co nejpozději.

Pracovníci s rozhodovací pravomocí

Rozhodovací pravomoci mají mít pracovníci, na nižších úrovních. Vývojáři musí chápat, jak jejich práce přispívá k celkovému cíli.

Uspokojovat požadavky zákazníků

Cílem je uspokojení současných a budoucích požadavků zákazníka, které se neustále mění v čase.

Zavést zpětnou vazbu

Pokud není možné specifikovat požadavky na začátku, je nutné zavést zpětnou vazbu a doplňovat je postupně. To ale znamená, provádět změny v průběhu (přepsat již hotový kód, upravit hotovou architekturu, předělat hotové rozhraní).

Odstranit lokální optimalizaci

Neměli bychom zbytečně plýtvat energií dílčích optimalizací, které nepovedou ke zvýšení efektivity.

Partnerství s dodavateli

Nejdůležitější je uspokojit potřeby zákazníka. Pro zrychlení a zlepšení kvality SW můžeme nakupovat hotové komponenty.

Zavést kulturu pro neustálé zlepšování

Zlepšit prostředí a motivace lidí, pro neustálé zlepšování procesů při vývoji SW.

2.7 Feature Driven Development

Metodika Feature Driven Development FDD vznikla v 90. letech minulého stolení Jeffem De Lucou a Peterem Coadem. FDD se řadí mezi agilní metodiky. Vývoj je řízen vlastnostmi „feature“, které jsou užitečné pro zákazníka. Vývoj probíhá v krátkých iteracích a realizace jedné vlastnosti by měla trvat typicky jednu dvoutýdenní iteraci. V rámci iterací se provádí činnosti návrhu a realizace jednotlivých vlastností. Metodika je více formální a také zdůrazňuje nutnosti modelování.

2.7.1 Základní praktiky podle FDD

Metodika definuje základní praktiky, které kombinací a jejich dodržováním vede k efektivnějšímu vývoji SW.

Doménové objektové modelování (Domain Object Modeling)

Objektový model domény poskytuje celkový rámec, do kterého se přidávají jednotlivé vlastnosti. Doménové objektové modelování se skládá z vytváření diagramů tříd (class diagrams) znázorňující objekty uvnitř jedné domény a jejich vztahy. Obvykle doplněné o sekvenční diagramy.

Vývoj podle vlastností (Developing by Feature)

Požadavky jsou definování jako vlastnosti „features“, které řídí další vývoj. Vlastnost je malá část funkčnosti, která je užitečná pro zákazníka. FDD definuje formální definici ve formátu <akce> <předmět> <podrobnosti>.

Individuální vlastnictví tříd (Individual Class Ownership)

FDD prosazuje individuální vlastnictví kódu. Vlastník zodpovídá za implementaci a testování. Protože daný kód zná je schopen rychle rozšířit a upravit danou třídu.

Týmy sestavované na základě vlastností (Feature Teams)

Vývoj probíhá podle vlastností a jednotlivé vlastnosti vyžadují většinou spolupráci více tříd tedy vlastníků, a proto je vytvořen tým, který dočasně sdružuje vlastníky tříd.

Inspekce (Inspections)

Úkolem inspekcí je odhalení defektů.

Pravidelné buildy (Regular Builds)

Pro odhalení integračních problému, se vytvářejí pravidelné buildy již hotových vlastností do jednoho celku.

Řízení konfigurací (Configuration Management)

Předmětem konfiguračního řízení je nejen zdrojový kód, ale také specifikace pažadavků, analytické a návrhové artefakty, testovací případy.

Reporting/viditelnost výsledků (Reporting/Visibility of Results)

Sběr informací o stavu projektu, provádět pokud možno automatizovaným a jednoduchým způsobem. Vyvarovat se porad o projektu a neproduktivní komunikace.

2.7.2 Fáze vývoje

Na obrázku č. 4 je znázorněno 5 fází vývoje podle metodiky FDD.

Vytvoření celkového modelu (Develop an Overall Model)

Cílem je nastínit základní a obecný účel systému. Model je často reprezentován diagramem tříd v jazyce UML. Součástí modelu může být předběžný seznam požadovaných vlastností.

Vytvoření seznamu vlastností (Build a Features List)

V této fázi se vytváří vyčerpávající seznam vlastností, měl by pokrývat co nejvíce požadavků na systém. Tyto vlastnosti rozdělit do skupin tak, aby vzájemně a logicky souvisely. Dále by v této fázi měl zákazník definovat klíčové vlastnosti vyvíjeného produktu.

Plánování (Plan by Feature)

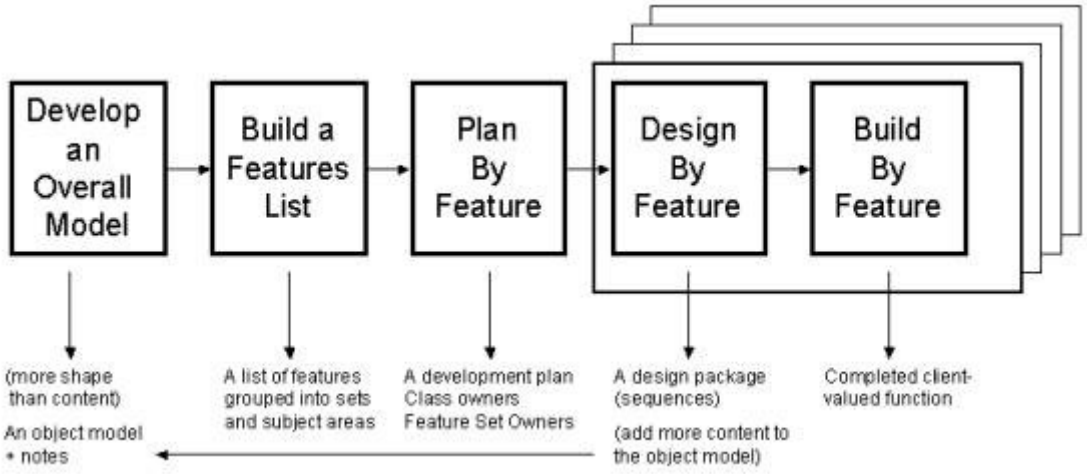
V této fázi se vytvoří plánovací tým, stanoví se datum ukončení vývoje, které by mělo být neměnné. Programátorům se přiřazují skupiny tříd, za které budou zodpovědní. Dále se vytváří předběžné pořadí, v jakém budou vlastnosti implementovány na základě priorit.

Návrh (Design by Feature)

Tato fáze se iterativně opakuje. Hlavní programátor vybere vlastnosti, které se v následující iteraci budou implementovat, dále určí třídy, které pokrývají danou vlastnost, a informuje vlastníky tříd, kteří vypracují detailní návrh.

Implementace (Build by Feature)

Každý vlastník zodpovídá za implementaci daných tříd, vytváří testovací případy a provádí jednotkové testy. Po provedení testů je třída vložena do sdíleného systému pro správu tříd.



Obrázek 5: Vývojové fáze FDD [13]

2.8 Adaptive Software Development

Adaptivní vývoj software (ASD) patří do agilních metodik, autorem je Jim Highsmith. Jak napovídá název „adaptive“ v metodice znamená, že průběžným změnám se nesmíme bránit, ale adaptovat je. Metodika je označována jako lehká. Statický životní cyklus plánování, návrh a realizace byl nahrazen dynamickým cyklem skládajícím se z fáze spekulace, spolupráce a učení.

2.8.1 Fáze životního cyklu

Spekulace

V této fázi dochází k plánování, ale jak už název naznačuje, spekulace dává větší prostor ke změnám. Při této fázi se stanovuje termín ukončení projektu, počet iterací, termín ukončení jednotlivých iterací, předběžné přiřazení komponent jednotlivým iteracím, rozhodnutí o použitých technologiích.

Spolupráce

V této fázi dochází k vývoji komponent, které můžou probíhat paralelně. Pro vytvořili fungujícího systému, týmy musí spolupracovat.

Učení

Fáze je zaměřena na zhodnocení dosavadního pokroku, zlepšování vývojového procesu a získávání ponaučení z chyb i úspěchů do dalších iterací.

2.9 Dynamic Software Development Method

Metodika Dynamic Software Development Method (DSDM) vznikla v 90. letech minulého stolení britským konsorciem DSDM, které se stará o rozvoj a její rozšiřování. Nejedná se pouze o metodiku, ale o framework obsahující podpůrné nástroje. DSDM patří do agilních metodik, využívá iterativní a inkrementální vývoj SW. Nevýhodou DSDM je, že se jedná o komerční produkt, je potřeba licence pro podpůrné nástroje, šablony atd.

2.9.1 Základní principy

Zaměření na obchodní potřeby

Tým se musí zaměřit na potřeby, které mají hodnotu pro zákazníka.

Časté dodávky

Časté dodávky jsou velmi přínosné pro úspěch projektu. Zrychluje se zpětná vazba zákazníka na projekt.

Spolupráce

Spolupráce je důležitá mezi členy týmu vede k rychlejšímu porozumění a vývoji, ale také tým by měl aktivně zapojit zákazníka.

Dodržování kvality

Začít testovat včas a průběžně, to znamená během celého životního cyklu. Včasné odhalení stojí méně nákladů na opravu a zvyšuje kvalitu SW.

Inkrementální vývoj

Systém je lepší dodávat po částech než celý na jednou. Postupné dodávky zvyšují důvěru zúčastněných stran a je zdrojem zpětné vazby, tým stále ověřuje správnost řešení.

Iterativní vývoj

DSDM využívá iterativní vývoj a iterativní dodávky, aby se změnami mohl dodávat lepší řešení. Tým by měl během každé iterace získávat zpětnou vazbu, nebránit se změnám.

Komunikace

Špatná komunikace často způsobuje selhání projektu. Týmy by měli provádět denní schůzky ve stoje, použít bohaté komunikační techniky jako jsou modelování a prototypování, vytvářet štíhlou, ale včasnou dokumentaci a podporovat neformální komunikaci mezi členy týmu.

Prokázání Kontroly

Po celou dobu projektu je nezbytná kontrola. Aktivně sledovat a řídit pokrok, plánovat pokrok, aby byl viditelný pro všechny, vyhodnocovat pokračování projektu na základě obchodních cílů.

2.9.2 Fáze životního cyklu

Životní cyklus podle DSDM se skládá ze 7 částí, jak je vidět na obrázku č. 7. V

Předprojektová fáze (Pre-Project)

Slouží k ověření, zda jsou všechny potřebné procesy zahájeny a správně nastaveny.

Studie proveditelnosti (Feasibility)

Klíčovou věcí v této fázi je stanovení rozsahu práce, kým, kdy a kde, stejně jako posouzení zda je DSDM vhodné pro daný projekt.

Základní fáze (Foundations)

Iterativní fáze. Zaměření na obchodní potřeby, získání klíčových požadavků a jejich priority, zjištění procesů jaké využívá zákazník, navrhnout architekturu řešení, identifikovat a hodnotit rizika projektu.

Zkoumání (Exploration)

Iterativní fáze. Slouží k vytvoření a zdokonalení obchodního modelu aplikace a k podrobnému definování požadavků na vyvíjený systém, dále se vytváří prototyp.

Vývoj (Engineering)

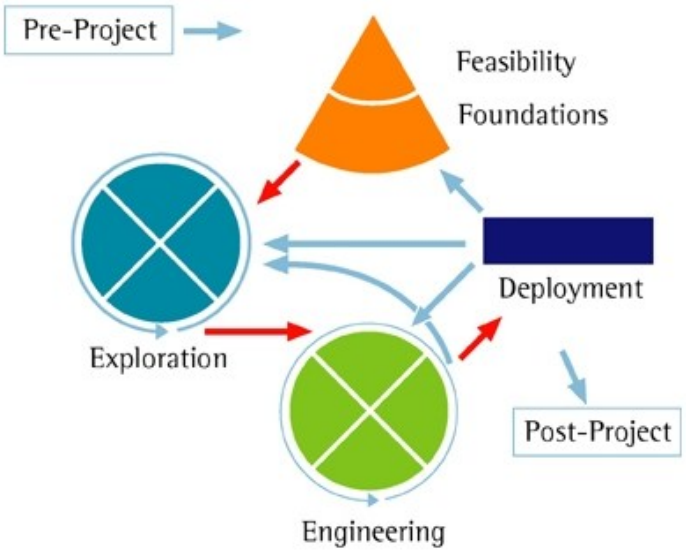
Iterativní fáze. Vývoj a implementace probíhá inkrementálně.

Nasazení (Deployment)

Primárním účelem je nasazení hotového produktu u zákazníka, případně školení uživatelů, nebo poskytnutí dokumentace a podpory. Tato fáze může také probíhat v několika iteracích v závislosti, zda zákazník požaduje průběžnou dodávku systému nebo celý systém na jednou.

Závěr projektu (Pos-Project)

Po posledním nasazení řešení u zákazníka. Cílem je posouzení, zda byly splněny přínosy popsané v obchodních případech.



Obrázek 6: DSDM životní cyklus

2.9.3 Role týmu

DSDM definuje role týmu, kterými jsou výkonný ředitel (sponzor), vizionář (visionary), projektový manažer (project manager), technický koordinátor (technical co-ordinator), vedoucí týmu (team leader), uživatelský poradce (business ambassador), analytik (analyst), vývojář (developer), tester (tester), moderátor (facillitator) a specialisté (specialist).

2.10 Microsoft Solutions Framework

Microsoft Solutions Framework (MSF) je obecný rámec pro vývoj sw, umožňuje vlastní implementaci nebo úpravu stávajících šablon metodik. První propracovaná část konkrétních praktik a návodů byla uveřejněna v roce 1993 vycházející ze zkušeností Microsoftu a partnerský firem.

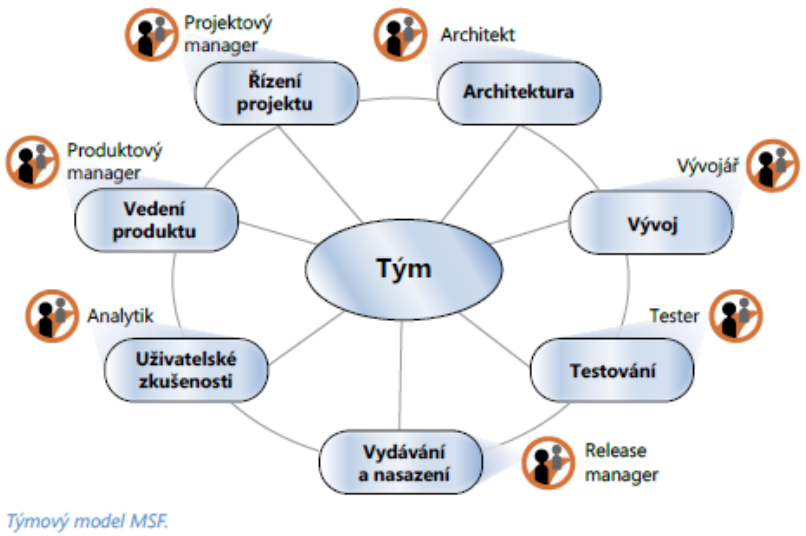
Součástí MSF jsou dvě metodiky MSF for Agile Software Development vycházející s agilního vývoje a druhá metodika MSF for CMMI Process Improvement splňující úroveň 3 CMMI modelu. MSF je spjaté s vývojovým prostředím Visual Studio Team Systém. Nabízí jednotnou a kompletní infrastrukturu pro návrh, vývoj, testování a nasazení SW aplikací. [18]

2.10.1 Základní principy

- Partnerství se zákazníkem
- Otevřená komunikace
- Pracovat na společné vizi
- Zapojení členů týmu
- Jasná zodpovědnost
- Soustředění se na obchodní hodnotu
- Adaptace na změny
- Investovat do kvality
- Učit se ze zkušeností
- Vždy vytvářet produkt, který je možno dodat a nasadit

2.10.2 Týmový model

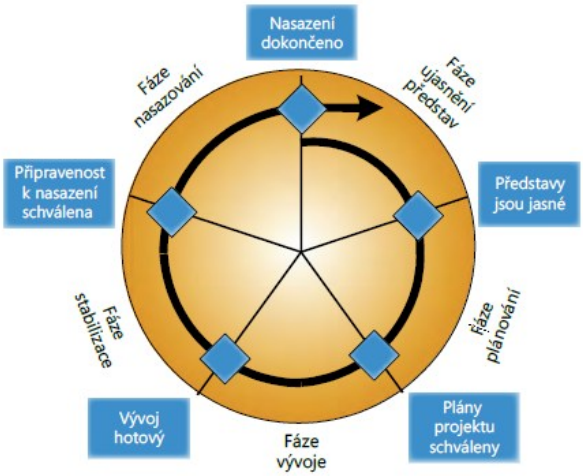
Popisuje přístup jak organizovat pracovníky a jejich aktivity s cílem vytvořit a dodat úspěšný projekt. Týmový model je rozdělen do 7 rovnocenných skupin podle funkcí a schopností lidí. Každá role odpovídá za kvalitu celkového řešení. Zavedené skupiny se můžou rozšiřovat nebo slučovat v závislosti na velikosti projektu, počtu lidí nebo na použité metodice (např. MSF for CMMI rozděluje týmový model na 17 skupin)



Obrázek 7: MSF - Týmový model [18]

2.10.3 Procesní model

Procesní model Microsoft Solution Framework můžeme popsat jako iterativní spirálový model opakující krátké vývojové cykly. Model se skládá z 5 fází a každá fáze končí milníkem, který slouží k ověření, zda jsme splnily to, co jsme si předsevzali. Po schválení můžeme pokračovat další fází.



Obrázek 8:MFS - Procesní model

3 Rozdělení metodik podle charakteristický znaků

Výběr vhodné metodiky pro vývoj SW projektu je komplikovaný, protože neexistuje jednotná klasifikace metodik pro jejich porovnání. Jednotlivé projekty se můžou lišit řadou aspektů, kterými jsou např. velikost projektu, schopnostmi a zkušenostmi manažera, schopnosti a motivace členů týmu, doménou projektu a dále podle využití projektu, typu aplikace a jiných vlastností projektu, týmu a podpůrných nástrojů.

Některé vlastnosti spolu úzce souvisí např. doba projektu je závislá na schopnostech a počtu lidí v týmu, ale naproti tomu větší počet lidí v týmu vyžaduje větší koordinaci a řízení. Stanovit na začátku projektu velikost a časové odhady pokud nemáme zkušenosti s tímto typem projektu a nemáme podrobnou specifikaci nebo případy užití, může být odhad dosti nepřesný.

Pro porovnání metodik charakterizuji jejich společné znaky, tedy kritéria klasifikace metodik. Inspiroval jsem se publikací od Aleny Buchalcecové [2][12], ve kterých se zabývá výběrem vhodné metodiky vývoje software. Nadefinoval jsem 9 kritérií a jejich stupnice ohodnocení, podle kterých v následující kapitole metodiky porovnám. U každého kritéria je stručný popis a výčet možných hodnot nebo stupnice hodnocení.

3.1 Popis kritérií pro porovnání

Popis devíti kritérií a jejich stupnice ohodnocení.

3.1.1 Kritérium Důležitost systému

Kritérium hodnotí závažnost dopadů při selhání systému.

Stupnice hodnocení

1 – 10: Čím větší hodnocení, tím je metodika vhodnější pro kritické systémy. (např 10. Selhávání systému může mít za následek ztrátu lidského života (řídicí systémy).

3.1.2 Kritérium Velikost projektu

Kritérium velikost projektu hodnotí vhodnost metodiky pro velké projekty.

Stupnice hodnocení

1 – 10: Čím větší hodnocení, tím je metodika vhodnější pro rozsáhlé projekty.

3.1.3 Kritérium Velikost týmu

Kritérium hodnotí velikost týmu na základě celkového počtu osob, které se účastní projektu.

Stupnice hodnocení

1: 1-4
2: 5-10
3: 11-20
4: 21-50

5: 51-100
6: Více než 100

3.1.4 Kritérium Délka iterací

Všechny moderní metodiky vyvíjí iterativním způsobem, proto kritérium hodnotí čas jedné iterace.

Stupnice hodnocení

1: 1-2 týdny
2: 2-4 týdny
3: Více než 1 měsíc
4: není iterativní vývoj

3.1.5 Kritérium Váha metodiky

Kritérium váha metodiky zahrnuje podrobnost, propracovanost a obsáhlost metodiky.

Stupnice hodnocení

1 – 10: hodnocení 0 znamená, že metodika je velmi lehká, nedefinuje např. role, artefakty, zodpovědnosti v projektu, ani se nezabývá řízením projektu. Obsahuje pouze obecná doporučení.

3.1.6 Kritérium Dostupnost uživatelů

Kritérium stanovuje, jak často musí být uživatel k dispozici týmu. Čím častěji se mění požadavky, tím by měl být uživatel více k dispozici.

Stupnice hodnocení

1: Uživatel je k dispozici pouze na začátku a na konci
2: Uživatel je k dispozici v předem stanovených bodech (specifikaci požadavků, milníky atd.)
3: Uživatel je k dispozici kdykoli na vyžádání
4: Uživatel je součástí týmu

3.1.7 Kritérium Neznalost domény

Kritérium hodnotí, zda je metodika vhodná pro projekt, kde s danou doménou projektu nemáme zkušenosti.

Stupnice hodnocení

1 – 10: Hodnocení 1 znamená, že metodika není vhodná pro projekt, kde nemáme žádné zkušenosti s danou doménou projektu.

3.1.8 Kritérium Stálost požadavků

Kritérium hodnotí, jestli metodika spíše předpokládá vydefinované požadavky na začátku nebo předpokládá, že požadavky se neustále budou měnit.

Stupnice hodnocení

1: Požadavky se neustále mění nelze je stanovit předem
2: Požadavky lze z velké části stanovit předem

3.1.9 Kritérium Vhodnost pro web projekty

Velkou část projektů tvoří internetové projekty, proto jsem zařadil, kritérium vhodnost pro internetové projekty, hodnotící jak je metodika vhodná a přizpůsobitelná webovým projektům.

Stupnice hodnocení

1: metodika nevhodná pro internetové projekty
2- 9: čím větší hodnocení tím více vhodná pro internetové projekty
10: metodika vytvořená speciálně pro internetové projekty

4 Porovnání metodik

Pro porovnání jsem vybral pět používaných metodik OpenUP, RUP, Scrum, FDD a XP. Na základě informací z online zdrojů a knih, které jsou v kapitole Literatura, jsem se snažil přiřadit kritériím odpovídající typické hodnoty.

	OpenUP	RUP	Scrum	FDD	XP
1. Důležitost systému	6	10	7	5	5
2. Velikost projektu	6	10	8	6	4
3. Velikost týmu	3	6	4	4	2
4. Délka iterace	2	3	2	2	1
5. Váha metodiky	7	10	5	7	5
6. Dostupnost uživatelů	2	2	4	2	4
7. Neznalost domény	8	9	7	7	7
8. Stálost požadavků	2	2	1	1	1
9. Vhodnost pro web p.	6	4	6	5	7

Tabulka 1:Ohodnocení metodik dle kritérií

4.1 Popis číselného ohodnocení metodik

Následující podkapitola se věnuje interpretaci předchozí tabulky s názvem Ohodnocení metodik dle kritérií a zdůvodněním přiřazeným hodnot jednotlivým metodikám.

Kritérium: Důležitost systému

Kritérium důležitost systému zohledňuje i přístup k testování. Kvalitu produktu podle RUP lze dělit podle požadavků FURPS, kde písmena představují: funkčnost (functional), použitelnost (usability), spolehlivost (reliability) výkon (performance) a podporu (supportability). Testování můžeme rozdělit jako black-box (uživatelské rozhraní) a White-box (testuje vnitřní strukturu např. unit testy). Pro důležité systémy, kde selhávání systému může způsobit ztrátu lidského života je vhodná metodika RUP.

Kritérium: Velikost projektu

Vhodnost metodiky pro velké projekty, teoreticky každá metodika může být použita pro velký projekt, ale záleží na čase dodání, jak bude dlouho projekt trvat. Z toho vyplývá, že metodiky s malým počtem lidí v týmu nejsou vhodné pro velké projekty. Nejméně vhodnou metodikou pro velké projekty je metodika XP a nejvíce vhodnou RUP.

Kritérium: Velikost týmu

Obecně lze říci, že agilní metodiky jsou vhodnější spíše pro menší týmy, ale metodika Scrum se zaměřuji i na komunikaci a řízení více menších týmů, které nemusí být v jedné

místnosti, budově, ale můžou být geograficky oddělené. Pro velké týmy je vhodná metodika RUP a Scrum. Pro malé týmy do 10 členů týmu, je vhodná metodika XP.

Kritérium: Délka iterace

Nejlépe vyhovující metodika pro velmi krátké iterace v rozmezí 1 – 2 týdny je XP. Naopak jako nejdelší čas iterace má přiřazena metodika RUP z důvodu použití metodiky na velkých projektech s velkým týmem a délka iterace je dána stanovením milníků. Typická délka iterace metodik Scrum a FDD je v rozmezí 2-4 týdnů. Délky jednotlivých iterací se můžou měnit.

Kritérium: Váha metodiky

Z hlediska propracovanosti a rozsáhlosti lze označit metodiku RUP jako nejvíce robustní metodiku, dále pak metodiku OpenUP spolu s metodikou FDD. Metodiky Scrum a XP jsou označovány jako lehké. Obě definují role týmu a artefakty. Scrum je zaměřen hlavně na oblast řízení projektu.

Kritérium: Dostupnost uživatelů

Z pohledu dostupnosti uživatelů, agilní metodiky XP a Scrum předpokládají úzké spojení s uživatelem nejlépe, aby byl součástí týmu. Metodika RUP, OpenUP a FDD zapojují uživatele nejvíce na začátku při sběru požadavků a potom ve stanovených bodech (např. milníky).

Kritérium: Neznalost domény

Kritérium neznalost domény hodnotí, jak je metodika vhodná pro projekty se, kterými jsme neměli žádnou zkušenost. V tomto ohodnocení zohledňuji techniky sběru požadavků a jejich správu, analýzu rizik a prioritizaci požadavků. Nejvíce vhodnou metodikou je RUP ohodnocenou nejvyšším počtem ze všech vybraných metodik.

Kritérium: Stálost požadavků

Pro toto kritérium jsem stanovil stupnici ohodnocení pouze se dvěma hodnotami, rozlišující základní rozdíl mezi rigorózními a agilními metodikami. Agilní metodiky předpokládají, že požadavky se neustále mění a nelze je stanovit všechny předem. Na rozdíl od rigorózní metodiky předpokládají stanovení požadavků předem s minimálními změnami. Proto metodika RUP a agilní metodika OpenUP (vycházející z UP a RUP) dostala přiřazeno hodnocení 2. Požadavky lze z velké části stanovit předem.

Kritérium: Vhodnost pro web. projekty

U internetových projektů záleží na včasném dodání. Vývoj probíhá iterativně, které je třeba doplnit o návrh grafického rozhraní a vzhledu stránek. Z hlediska vhodnosti pro internetové projekty jsem ohodnotil metodu XP jako nejvíce vhodnou z důvodu krátkých iterací a tedy rychlé zpětné vazby od zákazníka a nejméně vhodnou metodiku RUP.

5 Vytvoření metodiky a použití kombinace metodik na konkrétních modelových příkladech

Cílem této kapitoly je nastínit techniky použitelné při vytváření nového vývojového procesu nebo doplnění do stávajícího.

5.1 Doporučené techniky pro vytvoření metodiky a zlepšení vývojového procesu

Už při prvotní komunikaci zákazníka a zástupce vývojové firmy a domlouvání se na podmínkách projektu, musí být rozhodnuto, jak často bude zákazník spolupracovat s týmem, jestli bude ochotný přistoupit na agilní vývoj tedy s průběžným prezentováním pokroku a dodáváním částí systému, kde se může manipulovat s proměnnými šíře zadání, časem a náklady anebo na vývoj klasický. Na základě tohoto rozhodnutí se musí přizpůsobit i samotná smlouva. Při klasickém vývoji bude obsahovat podrobnou specifikaci a funkce, které musí být implementovány, aby zadání mohlo být prohlášeno za splněné.

Vývoj v každém případě provádět inkrementálním a iterativním způsobem a snažit se co nejvíce zapojit zákazníka. V první části odstartování projektu je potřeba nastínit cíle projektu, obchodní přínos a zachytit hlavní požadavky systému. Utváří se celková vize projektu, plán projektu, který se může měnit na základě iterací a změn funkčnosti a požadavků zákazníka. Sběr požadavků je velmi důležitý u všech projektů. Základním rozdělení požadavků je na funkční a nefunkční. Funkční požadavky popisují požadované vlastnosti a služby. Nefunkční požadavky mohou zahrnovat různá omezení na provedení, design, rychlost odezvy, aj. Dalším úkolem je stanovit prioritu požadavků a analyzovat možná rizika spojená s vývojem (např. neznalost vyvíjené technologie).

5.1.1 Sběr a hledání požadavků

Požadavky vyplívají z kontextu systému (přímí uživatelé systému, zainteresované osoby, další systémy, obchodní cíle aj.). Pro úspěšný sběr požadavků můžeme zvolit různé techniky.

Skupinové konzultace

Obvykle je lepší vést pohovory s jednotlivými zainteresovanými lidmi zvlášť.

Rozhovory s uživateli

Nejlepším způsobem sběru požadavků jsou právě osobní rozhovory. Otázky mají být kladené tak, aby uživatele rozpovídala a my se dozvěděli, co od systému očekávají a potřebují.

Dotazníky

Dotazníky nenahradí osobní rozhovory, můžou být užitečným doplněním osobních pohovorů, ze kterých jsme si vytvořili podklady pro doplňující otázky, které může aplikovat na velký počet uživatelů systému. Otázky z velké části formulovat tak, aby odpověď byli typu ano/ne nebo připraveny možnosti, které uživatel může vybrat.

Brainstorming a Workshop požadavků (Dílna požadavků)

Jsou to efektivní techniky pro zjišťování požadavků. Princip je ve spontánní diskuzi na téma hledání nápadů. Všechny nápady jsou přijímány jako dobré a nevede se o nich diskuze. Jsou zapisovány a po skončení jsou analyzovány.

Analýza uživatelských úkolů

Další možností hledání a sběru požadavků je analýza uživatelských úkolů.

Uživatelské příběhy (User Story)

Každý požadavek se rozepíše jako tzv. uživatelský příběh. Je psán jazykem zákazníka.

5.1.2 Plánování a sledování pokroku

Provádět Release Planning(plánování uvolnění verze) a Iteration planning(plány na jednotlivé iterace). Z metodiky Scrum jsem se inspiroval metodou plánování nad požadavky sepsanými v produktovém backlogu ze kterých zákazník nebo jeho zástupce vybere spolu s týmem položky, které se budou realizovat v rámci sprintu, na základě odhadu kolik jednotek se splní v rámci sprintu a jednotek přiřazeným položkám. Produktový backlog může být psát v tabulkovém editoru Excel.

Jednotlivé úkoly v rámci iterace si mohou vybrat programátoři sami nebo je přiděluje hlavní programátor, který sleduje jejich plnění a pokrok. Pro sledování stavu je vhodné použít krátké každodenní schůzky v pevně zvolený čas. Na kterých by každý řekne co dělal včera, co bude dělat dnes a jestli má problémy nebo překážky, které mu brání ve splnění úkolu.

5.1.3 Modelování

Pro modelování systémů využít modelovací jazyk UML, který je standardizován. Před zběsilým vytvářením diagramů se vždy rozhodnout, jestli má pro vývojový tým nebo pro zákazníka nějaký smysluplný přínos a do jakých podrobností modelovat z důvodu neplýtvání zdrojů.

Základní typy diagramů UML:

- Třídní diagram
- Objektový diagram
- Diagram balíčků
- Diagram komponent
- Strukturální diagram
- Diagram nasazení
- Diagram případů užití
- Diagram aktivit
- Stavový diagram
- Sekvenční diagram
- Diagram komunikace
- Diagram přehledu interakcí

5.1.4 Programátorské techniky

Pokud se předpokládá časté sdílení kódu, v rámci týmu. Měli by si programátoři stanovit společný standarty psaní zdrojového kódu (rozvržení bloků if-else, jak pojmenovávat proměnné, konstanty, funkce nebo třídy ad.).

Při zapojení nového méně zkušeného programátora do již rozběhnutého projektu bych doporučil zavést párové programování, kdy nový programátor se rychle zapracuje, naučí se používané standarty kódování, získá přehled o již naimplementových částech a odpadá kontrola vytvořeného kódu.

Pokud nemáme automatizované testy, tak refaktorování zdrojového kódu může být obtížné respektive, může způsobit zavlečení dalších chyb do programu.

5.1.5 Testování

Testování projektu je také velmi důležité, oprava objevené chyby až u zákazníka stojí více času a peněž než v průběhu vývoje a navíc chyby na zákazníka působí nekvalitně odvedenou prací a odrazuje to od další spolupráce s námi. Jak již bylo zmíněno základní rozdělení je na black-box a white box. Míra testování bude záležet také na důležitosti, typu projektu, finančních aspektech a počtech testerů. Můžeme pro automatizaci použíti testovací nátroje např. jUnit, Microsoft Visual Studio Test Professional, IBM Rational Tester a mnoho dalších.

Typy testů:

- Jednotkové Testování (Unit testing)
- Integrační testování
- Akceptační testování
- Funkční testování
- Nefunkční testování
- Manuální testování
- Regresní testování
- Testy uživatelského rozhraní

5.2 Kombinace metodik na modelových příkladech

V této části se pokusím nastínit popis vývojového procesu na konkrétních příkladech. Jako příklad jsem vybral implementaci informační systém pro autoškolu a jako druhý příklad vytvoření pokladního systému. Veškeré informace o projektech jsou vymyšlené.

5.2.1 Příklad: Informační systém autoškoly

Příklad má demonstrovat vývoj podle FDD a klasických metod. Zákazník je majitel autoškoly, který chce vytvořit nový informační systém z důvodu zkvalitnění služeb pro žáky, učitele a vlastní potřeby evidence.

Od dodaného software očekává možnosti evidence studentů, zaměstnanců a jejich skupin, které můžou učit, vozidel autoškoly, studentský přihlášek na jednotlivé řidičské

skupiny, seznamy komisařů, kteří budou přítomni u závěrečných zkoušek. Dále pak očekává možnosti přihlašování studentů na jednotlivé jízdy a zkoušky, plánování rozvrhu jízd. Software musí splňovat potřebné podklady pro vedení účetnictví (seznamy plateb kurzů, doplňkových výukových jízd, opakování zkoušek, nákladů na provoz aut …).

Předpokládám, že zákazník osloví malou firmu vyvíjející software na zakázku a souhlasí s dodáním celého informačního systému naráz s průběžnou kontrolou a demonstrací již hotových částí.

V první části je potřeba hlouběji porozumět vyvíjenému systému. Zákazník se pokusí popsat v článku, proč chce nový informační systém, vysvětlí očekávané přínosy systému, jaké funkce a vlastnosti by měl umět. Detailnější získávání požadavků bude prováděno schůzky a rozhovory. Zjistit, kdo se systém bude pracovat a jaké vlastnosti a funkce bude moci v systému provádět. Vytvořit případy užití, které se dále podrobněji rozpracují, vytvořit prvotní plán projektu, stanovit délku iterací. Na základě identifikovaných entit vytvořit datový slovník.

V další části probíhá návrh architektury systému, spolu se zákazníkem stanovují priority jednotlivých funkcí a vytváří se plán pro nejbližší iteraci. Dále se rozkreslují případy užití, vytváří návrh grafického uživatelského rozhraní. Do iterace se vyberou jednotlivé funkce, programátoři začínají s implementací.

Následně v iteracích probíhá plánování a vybírání funkcí, které programátoři implementují a otestují. Na konci každé iterace dojde k předvedení již hotových částí. Budujeme si tak důvěru a zákazník, má možnost reagovat na současný stav systému a případně si mohl vzpomenout na nový požadavek nebo úpravu stávajícího. Dochází k neustálé integraci implementovaných částí. Testeři píší testovací scénáře, vytvářejí automatické testy uživatelského rozhraní a provádí i testy manuální. V průběhu vývoje dokumentarista vytváří uživatelskou dokumentaci.

Jakmile jsou všechny funkce systému implementované a otestované a zákazník je spokojen se stavem. Dojde k ukončení vývoje a systém je nasazen u zákazníka. Následně jsou opravovány chyby objevené za provozu.

5.2.2 Příklad: Pokladní systém propojený se skladovým systémem

Příklad má demonstrovat průběh vývoje řízeného pomocí metodiky Scrum . Zákazník požaduje vytvoření pokladní systém ovládaný s pomocí klávesnici nebo dotykově, tato pokladny budou propojeny se skladovým systémem.

Při vývoji je součástí product owner (vlastník produktu) jehož cílem je porozumění produktu, požadavkům zákazníka. Vytváří celkovou vizi projektu. Při komunikaci s týmem zastupuje zájmy zákazníka, neřídí jednotlivé členy týmu, pouze stanovuje priority, co se má dělat a v jakém pořadí. Typově by to měl být člověk komunikativní s analytickým myšlením.

Průběh vývoje začíná komunikací zákazníka a product ownera, zjištěním požadavků a vlastností kladeným na systém, které společně sepíší s využitím běžného jazyka zákazníka a vytvoří uživatelských příběhy (user story).

Příklady uživatelského příběhů pro sklad:

- Jako pracovník skladu (skladník) potřebuji naskladnit zboží na určená skladová místa.

- Jako skladník potřebuji zobrazit disponibilitu jednotlivého zboží, z důvodu inventury.
- Vedoucí potřebuje mít možnost editace platnosti, dostupnosti a daní výrobků poznámka celkových vlastností výrobků.
- Při požadavku na vyskladnění potřebuji vytisknout výdejku.

Příklady uživatelského příběhů pro pokladnu:

- Se systémem můžou pracovat pouze autorizované osoby
- Při pokladním prodeji potřebuji zjistit počet dostupného zboží na skladě.
- Jako vedoucí prodeje potřebuji jednotlivým zaměstnancům přiřazovat oprávnění, co mohou a nemohou provádět za operace.
- Při změně pokladní se provede výčetka, peněz v pokladním boxi a provede se úzávěrka směny (z-report).
- Pokladní potřebuje mít možnost uzamknout pokladní systém, aby nikdo nemohl pracovat na dané pokladně.
- Vratky zboží může provádět pouze oprávněné osoba.

Vlastník produktu získané požadavky zanalyzuje, rozepíše na jednotlivé úlohy (tasky), určí prioritu s přínosem pro zákazníka. Vlastník produktu je sepíše do tzv. product backlogu. Obsahuje jednoznačný identifikátor, jméno, důležitost, odhad délky trvání, skutečnou délka trvání a případně poznámku např. že je potřeba vytvořit sekvenční diagram. Backlog se v průběhu vývoje neustále vyvíjí a doplňuje.

V další části přichází na řadu plánování sprintu.. Produktový vlastník nastíní cíl sprintu, ale ohodnocení jednotlivých úkolů je v kompetenci týmu. Z produktového backlogu se vyberou ty části, které odhaduje tým zvládnout v rámci pevně stanovené délky sprintu. Pro ohodnocení se využívá hra plánovacího pokeru. Každý člen týmu má karty ohodnocené např. 0, 0.5, 1, 2, 3 , 5, 8, 13, 20, 40, 100, otazník a přestávka. Nula znamená, že neumí odhadnout, přestávka znamená únavu s plánování a nekonečno znamená návrh rozdělit úkol na menší. Při velkém rozptylu se vytváří diskuze, aby se shodli na ohodnocení.

Po skončení plánování sprintu následuje vývoj. Vývojový tým je samoorganizující to znamená, že členové týmu, si sami vybírají položky ze sprint backlogu. Pro sledování pokroku v rámci sprintu se provádí každodenní schůzky na, kterých každý člen týmu řekne, co dělal včera, jestli má nějaký problém a co bude dělat dnes. Na konci sprintu se provede zhodnocení co je ukončené, skoro hotové a nezačaté. Dále se předvede potenciálně hotová verze. Následně pokud produkt backlog obsahuje nějaké položky, plánuje se další sprint.

6 Doporučení pro modelové případy vývoje software

V této kapitole popíši vytvořený program, sloužící pro doporučení vhodné metodiky vývoje SW na základě vlastností nového projektu. Podrobná uživatelská příručka s popisem jednotlivých obrazovek a funkcí se nachází v příloze [B].

Jedná se o desktopovou aplikaci s názvem Výběr vhodné metodiky. Je implementována ve vývojovém prostředí Microsoft Visual Studio 2010 s využitím technologie Windows Presentation Foundation (WPF) a programovacího jazyka C#. Technologie WPF umožňuje vytvářet graficky bohaté aplikace, které mají oddělený vzhled od logiky aplikace. Pro ukládání dat jsem využil lokální databázi SQL Server Compact Edition.

Pro znázornění chování programu z pohledu uživatele, jsem vytvořil model případu užití (Use-Case) v programu v Microsoft Visio 2010.

Program obsahuje 5 předdefinovaných vývojových metodik s ohodnocením kritérií. Tuto databázovou základnu metodik lze pomocí základních funkcí systému rozšířit anebo upravit, z důvodu subjektivního ohodnocení kritérií. Dalšími požadavky na systém bylo, aby uživatel měl možnost nastavovat preference kritérií. Toto nastavení může výrazně ovlivnit výběr doporučené metodiky vývoje SW. Hlavní funkcí je vyhodnotit ohodnocení kritérií nového projektu se všemi uloženými metodikami a doporučit nejvhodnější metodiku.

Následující 2 podkapitoly popisují způsob stanovení vah kritérií a princip algoritmu doporučení metodiky.

6.1 Stanovení vah kritérií

Pro stanovení vah kritérií existuje více metod např. Metoda párového srovnání, Fullerova metoda, Bodovací metoda a Saatyho metoda. V programu jsem zvolil pro stanovení vah kritérií Bodovací metodu. Vybral jsem ji na základě její jednoduchosti a rozhodovatel je schopen určit nejenom pořadí, ale i poměr důležitosti kritérií.

Princip Bodovací metody

Všechna kritéria se ohodnotí počtem bodů ze stanovené stupnice 0 – 100. Čím více bodů je přiřazeno, tím je kritérium významnější. Váha kritéria se vypočítá jako podíl počtu přiřazených bodů danému kritériu a sumě všech přiřazených bodů.

Váha kritéria může výrazně ovlivnit výběr vhodné metodiky pro projekt. V tabulce č. x je ukázka bodování a určení váhy kritéria, příklad nejvíce preferuje vhodnost pro webové aplikace délka iterace, dostupnost uživatelů a nejméně se preferuje znalost domény (protože např. doména je známá).

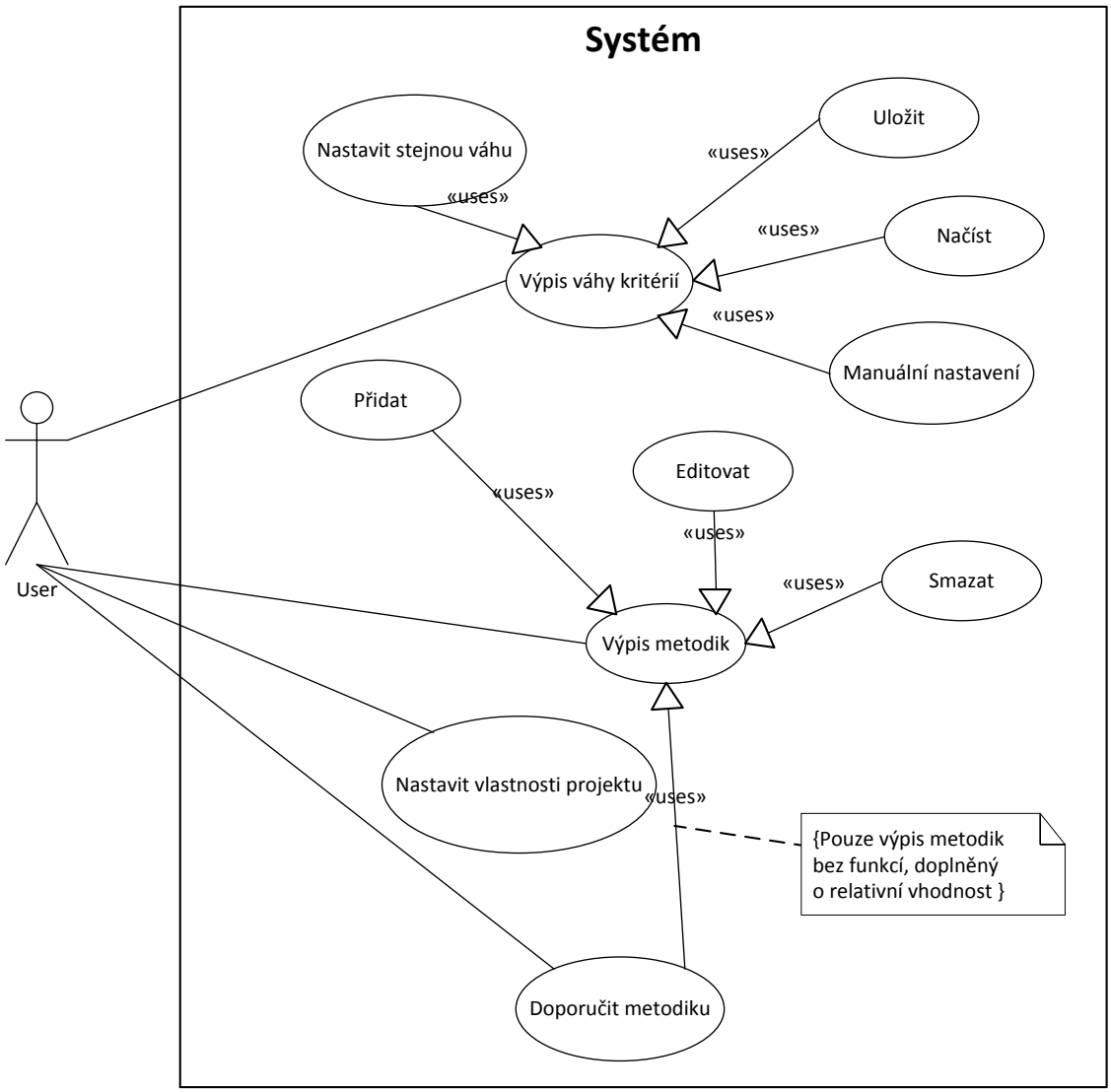
Kritéria	Body	Váha
1. Důležitost systému	30	0,058
2. Velikost projektu	50	0,097
3. Velikost týmu	63	0,123
4. Délka iterace	85	0,166
5. Váha metodiky	40	0,078
6. Dostupnost uživatelů	75	0,146
7. Neznalost domény	15	0,029
8. Stálost požadavků	70	0,136
9. Vhodnost pro web projekty	85	0,166
	513	1

Tabulka 2: Stanovení vah kritérií

6.2 Princip algoritmu doporučení metodiky

Algoritmus je založen na výběru metodiky, která je nejblíže ohodnocením kritérií k novému projektu.

- Prvním krokem je pro všechny metodiky spočítat absolutní hodnotu rozdílu kritéria nového projektu s hodnotou kritéria dané metodiky.
- Druhým krokem je převod na normovaný tvar, vypočítaný jako podíl výsledku z předchozího kroku s maximální hodnotou stupnice kritéria.
- Třetím krokem je aplikování váhy kritéria tzn. součin výsledku z předchozího kroku s vypočítanou váhou kritéria na základě bodovací metody.
- Čtvrtým krokem je výpočet relativní vhodnosti, vypočítaný jako suma výsledků z 3. kroku pro každou metodiku.
- Posledním krokem je setřídění výsledků relativní vhodnosti od nejlépe vyhovujících metodik (nejblíže 0) po nejméně vhodné.



Obrázek 9: Diagram případů užití – Výběr vhodné metodiky

7 Závěr

V průběhu práce jsem se seznámil blíže s popsanými metodikami vývoje software, zjistil jsem, že existuje velké množství metodik vhodné pro různé typy a vlastnosti projektů. Agilní metodiky jsou si podobné, protože dodržují společné základy Manifestu agilního vývoje, hlavním cílem všech metodik je vytvářet software rychleji a efektivněji s uspokojením požadavkům zákazníka.

Z důvodu neexistence jednotné klasifikace, jsem část práce věnoval stanovením kritérií pro porovnání a následně vybrané metodiky porovnal. Dále jsem vytvořil software, pro pomoc s výběrem vhodné metodiky na základě vlastností nového projektu. V další části práce jsem popsal vhodné techniky pro zlepšení vývojového procesu, které vývojový tým může zkombinovat a vytvořit si tak vlastní metodiku vývoje vyhovující jeho potřebám. Dále jsem uvedl použití kombinace metodik na modelových příkladech.

Na závěr bych chtěl říct, že metodika je pouze nástroj, který nám pomáhá s vývojem, ale nezaručí úspěšnost každého projektu. Základem jsou spolupracujícími a komunikující lidé a jejich schopnostmi.

Námětem pro další vývoj projektu může být například zahrnutí do porovnání další metodiky vývoje a dále by mohlo být přínosné rozšíření kritérií pro porovnání z důvodu citlivějšího výběru metodiky.

8 Literatura

[1] KADLEC, Václav. *Agilní programování*: *metodiky efektivního vývoje softwaru*. 1. vyd. Brno: Computer Press, 2004, 278 s. ISBN 80-251-0342-0.

[2] BUCHALCEVOVÁ, Alena. *Metodiky budování informačních systémů*. Vyd. 1. Praha: Oeconomica, 2009, 205 s. Vysokoškolská učebnice (Oeconomica). ISBN 978-802-4515-403.

[3] BECK, Kent. *Extrémní programování*. Vyd. 1. Praha: Grada, 2002, 158 s. ISBN 80-247-0300-9.

[4] ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací*: *objektově orientovaná analýza a návrh prakticky*. Vyd. 1. Překlad Bogdan Kiszka. Brno: Computer Press, 2007, 567 s. ISBN 978-80-251-1503-9.

[5]*Agile Manifesto* [online]. 2001. Dostupné z: http://agilemanifesto.org/

[6] *Risk reduction with the RUP phase plan* [online]. Dostupné z: http://www.ibm.com/developerworks/rational/library/1826.html

[7] *Rational* [online] Dostupné z: www.rational.com

[8] *Extreme Programming* [online]. 1999-2009. Dostupné z: http://www.extremeprogramming.org/

[9] *Scrum Guide* [online]. únor 2010. Dostupné z: http://www.aguarra.cz/pdf/Scrum-Guide-Czech-Language.pdf

[10] *Scrum* [online]. 2005. Dostupné z: http://epf.eclipse.org/wikis/scrum/index.htm

[11] *Lean Development*: *Poppendieck.LLC* [online]. 2000-2010. Dostupné z: poppendieck.com

[12] BUCHALCEVOVÁ, Alena. *Metodiky vývoje a údržby informačních systémů*: *kategorizace, agilní metodiky, vzory pro návrh metodiky* [online]. 1. vyd. Praha: Grada, 2005, 163 s. [cit. 2012-02-05]. ISBN 80-247-1075-7. Dostupné z: http://nb.vse.cz/~buchalc/clanky/metodiky.pdf

[13] PALMER, Stephen R a John M FELSING. *A practical guide to feature-driven development*. Upper Saddle River, NJ: Prentice Hall PTR, 2002, 271 s. ISBN 01-306-7615-2.

[14] *Cockburn Alistair* [online]. 1999. Dostupné z: http://alistair.cockburn.us/Methodology+per+project

[15] *Open UP Papers* [online]. 2007. Dostupné z: http://www.eclipse.org/epf/general/OpenUP.pdf

[16] EPF. *Open UP* [online]. 8. vyd. 30.11.201. Dostupné z: http://epf.eclipse.org/wikis/openup/

[17] DSDM CONSORTIUM. *DSDM* [online]. Dostupné z: http://www.dsdm.org

[18] MICROSOFT. *Správa IT služeb a řízení životního cyklu softwarových aplikací*. 2007. Dostupné z: http://download.microsoft.com/download/5/5/d/55da927f-d3fb-43dd-8a73-2bc2f96be56a/Zivotni_cyklus_aplikaci_FINAL.pdf

[19] FRIEBELOVÁ, Jana. *Vícektriteriální hodnocení variant*. Dostupné z: http://www2.ef.jcu.cz/~jfrieb/rmp/data/teorie_oa/VICEKRIT_HODNOCENI.pdf

[20] KORVINY, Petr. *Teoretické základy vícekriteriálního rozhodování*. Dostupné z: http://korviny.cz/mca7/soubory/teorie_mca.pdf

[21] KNIBERG, Written by Henrik. *Scrum and xp from the trenches*: *how we do scrum* [online]. S.l.: [C4Media Inc.], 2007 [cit. 2012-05-03]. ISBN 978-143-0322-641. Překlad dostupný z: http://www.infoq.com/resource/news/2007/06/scrum-xp-book/en/resources/Scrum%20and%20XP%20from%20the%20Trenches%20-%20Slovak.pdf

9 Seznam Příloh

Příloha A: Obsah přiloženého CD

Přiložené CD obsahuje:

- Popis obsahu přiloženého CD v hlavním adresáři Obsah.txt
- **Text** - Tento text diplomové práce obsahující kopií oficiálního zadání a prohlášení. (formát .docx, .pdf/A)
- **SourceCode** - Zdrojové kódy vytvořeného programu
- **Instal** - Instalační soubor programu výběr vhodné metodiky.

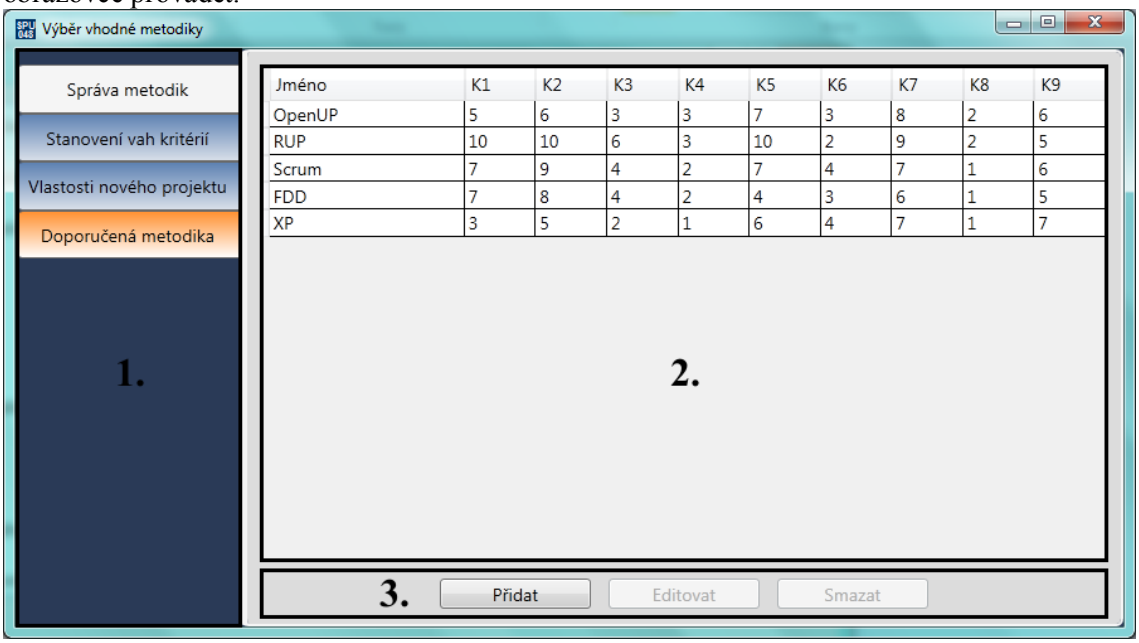
Příloha B: Uživatelská příručka

1. Úvodem

Program výběr vhodné metodiky slouží jako doporučení vhodné metodiky vývoje SW na základě stanovení vah kritérií a ohodnocení vlastností nového projektu.

2. Rozdělení okna programu

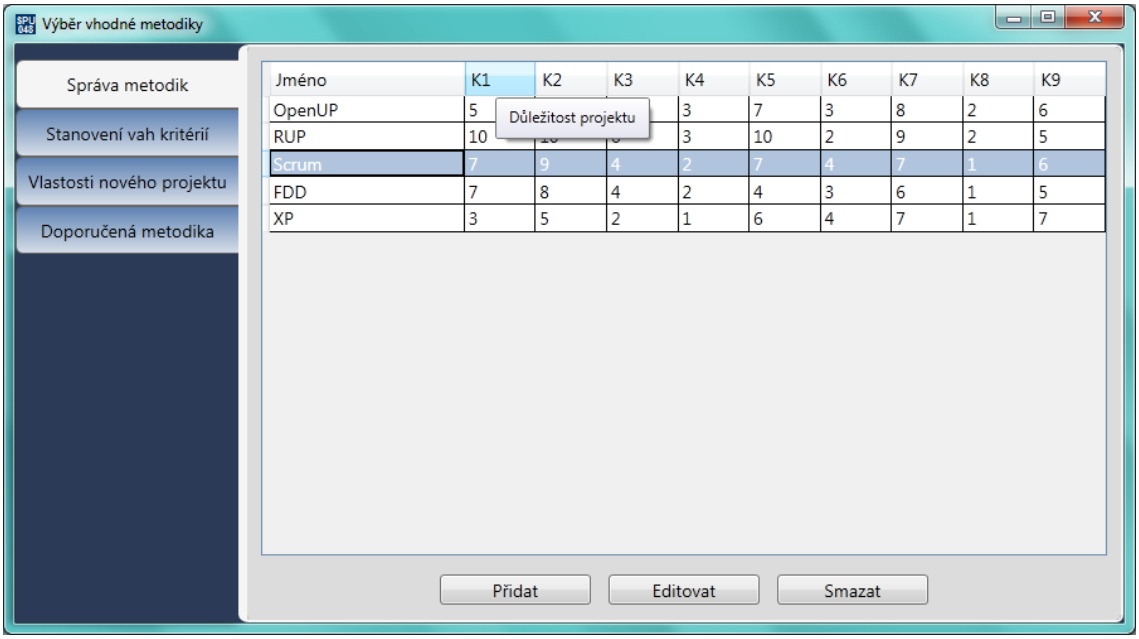
Okno programu Výběr vhodné metodiky je rozděleno na tři hlavní části. Na levé straně je menu označeno (1.) na obrázku níže. Pravá horní část slouží k zobrazení informací a obsahu a vybrané položky menu (2.) a poslední část (3.) pravá spodní obsahuje funkce, které lze na dané obrazovce provádět.



3. Správa metodik

V této položce menu získáte přehled o všech uložených metodikách v databázi s možností vytváření nových, editací a mazání stávajících.

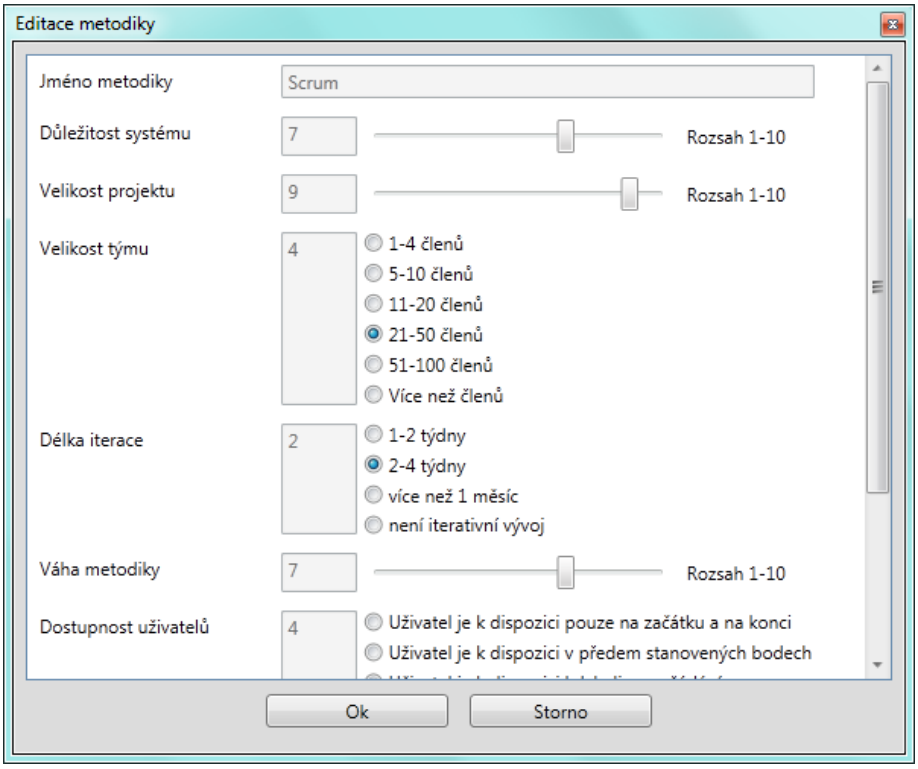
Oblast obsahu zobrazí všechny metodiky uložené v databázi v seznamu s názvy metodik a ohodnocením kritérií. Při najetí myši na hlavičku tabulky, např. na K1 se vám zobrazí textový popis, v tomto případě „Důležitost projektu“ a při kliknutí tabulku setřídíte.



Funkce Přidání a Editace metodiky

Při kliknutí na tlačítko Editovat dojde k zobrazení okna s názvem „Editace metodiky“. Pokud kritérium obsahuje výčet hodnot, změnu provedete výběrem nové hodnoty. Pokud kritérium obsahuje pouze rozsah, tak úprava provedete posunem jezdce.
Tlačítkem Ok potvrdíte změny, které se uloží do databáze, nebo tlačítkem Storno okno zrušíme.

Přidání metodiky se provádí kliknutím na tlačítko Přidat, ohodnocení probíhá stejně jako editace s tím rozdílem, že se vyplňuje název metodiky, který musí být unikátní.

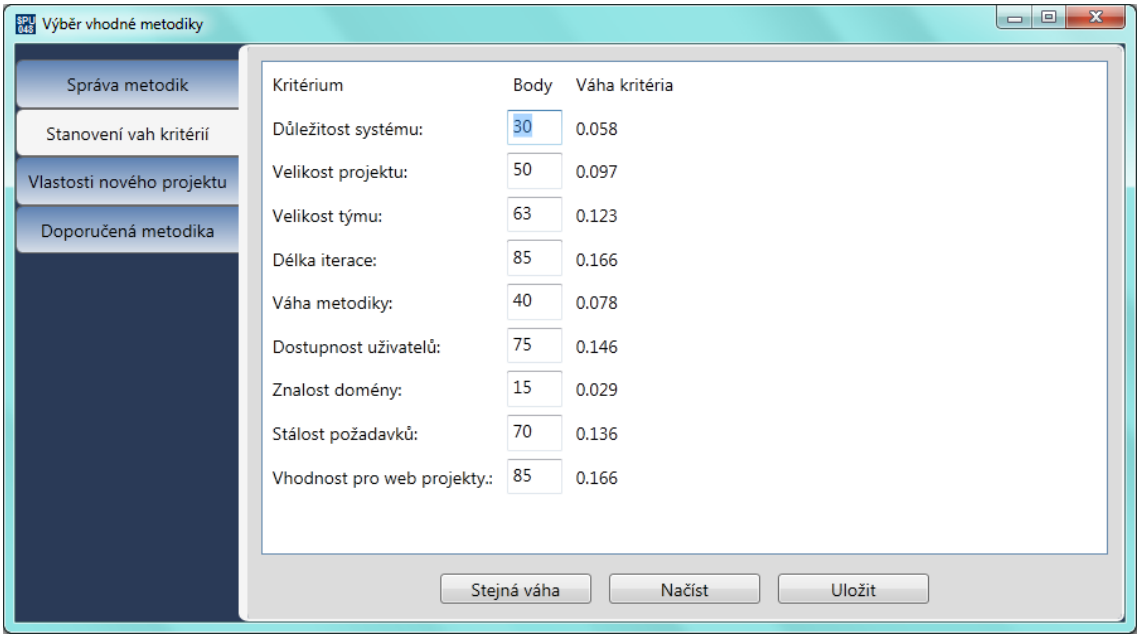


4. Stanovení vah kritérií

Váha kritéria se počítá dynamicky při změně bodů. Pokud chcete, aby dané kritérium nebylo zahrnuto do vyhodnocování, přiřaďte 0 bodů. Jednomu kritériu můžete přiřadit maximálně100 bodů.

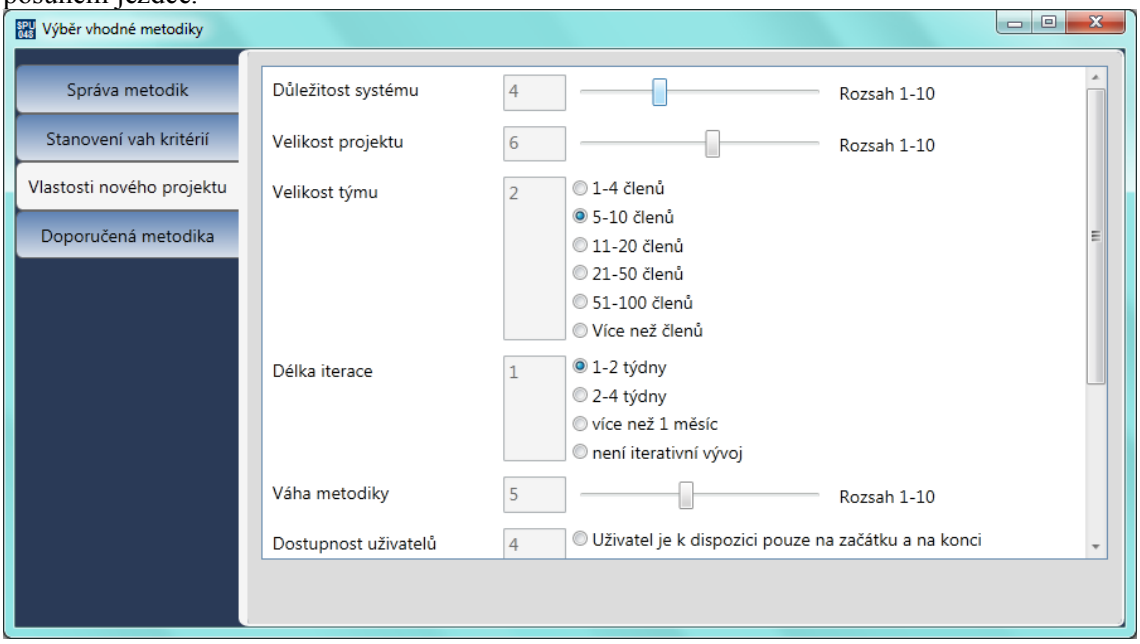
Funkce

Při kliknutí na tlačítko stejná váha, program přiřadí stejný počet bodů všem kritériím. Vaše bodové ohodnocení si můžete uložit anebo načíst poslední uloženou konfiguraci.



5. Vlastnosti nového projektu

Výběrem této položky menu se zobrazí seznam kritérií, která musíte ohodnotit. Defaultní hodnoty jsou nastaveny na 1. Pokud kritérium obsahuje výčet hodnot, změnu provedete výběrem nové hodnoty. Pokud kritérium obsahuje pouze rozsah, tak úprava provedete posunem jezdce.



6. Doporučená metodika

Po kliknutí na tlačítko vyhodnotit program spočítá relativní vhodnost všech dostupných metodik. Čím je relativní vhodnost bližší 0 tím, je metodika vhodnější pro náš modelový

projekt. Informativní okno zobrazí první 3 nejlépe vyhovující metodiky. Pod nimi je zobrazena tabulka se všemi, pro možnost analyzování, ve kterých kritériích se shoduje nový projekt s metodikami.